Inférence semi-symbolique

M2 MPRI - Langages de programmation probabilistes

Guillaume Baudart

Calculer exactement la distribution a posteriori décrite par un modèle est généralement un problème très difficile. Les méthodes d'inférences particulaires permettent d'approximer le résultat en rejouant le modèle de nombreuses fois de manière indépendante. Le calcul exact reste cependant possible pour les modèles les plus simples, ou pour certains sous-termes au sein d'un modèle complexe. Le but de ce projet est d'implémenter une méthode d'inférence semi-symbolique qui combine calcul symbolique exact et méthode particulaire approchée pour calculer la distribution a posteriori décrite par un modèle.

1 Préambule

Considérons les deux programmes suivants où `a`, `b`, `mu0`, `sigma0`, `sigma` sont des constantes.

```
let beta_bernoulli v =
  let p = sample (beta ~a ~b) in (* p ~ Beta(a, b) *)
  observe (bernoulli ~p) v;      (* v ~ Bernoulli(p) *)
  p

let gauss_gauss v =
  let mu = sample (gaussian ~mu:0. ~sigma:1.) in (* mu ~ N(0, 1) *)
  observe (gaussian ~mu:mu ~sigma:1.) v;         (* v ~ N(mu, 1) *)
  mu
```

Question 1. En utilisant la sémantique par noyaux vue en cours (cours 2), montrer que :

- $[\![\texttt{infer beta\_bernoulli } v]\!] = Beta(a+v,\ b+(1-v))$
- $[\![\texttt{infer gauss\_gauss } v]\!] = \mathcal{N}(m, s)$ avec $m = v/2$ et $s^2 = 1/2$

Plus généralement, si la distribution a posteriori est de la même famille que la distribution a priori après une observation, on dit que la distribution a priori et la distribution observée sont *conjugées*. On a ainsi :

- Si $\begin{cases} X \sim Beta(a,\ b) \\ Y \sim Bernoulli(X) \end{cases}$ alors $\begin{cases} Y \sim Bernoulli\left(\dfrac{a}{a+b}\right) & \textit{marginale} \\ (X \mid Y = v) \sim Beta(a+v,\ b+(1-v)) & \textit{conditionnement} \end{cases}$

- Si $\begin{cases} Y \sim \mathcal{N}(X, \sigma) \\ X \sim \mathcal{N}(\mu_0,\ \sigma_0) \end{cases}$ alors $\begin{cases} Y \sim \mathcal{N}(\mu_0, \sqrt{\sigma_0^2 + \sigma^2}) & \textit{marginale} \\ (X \mid Y = v) \sim \mathcal{N}(\mu_1,\ \sigma_1) & \textit{conditionnement} \end{cases}$

avec $\begin{cases} \mu_1 = \left(\dfrac{\mu_0}{\sigma_0^2} + \dfrac{v}{\sigma^2}\right) \Big/ \left(\dfrac{1}{\sigma_0^2} + \dfrac{1}{\sigma^2}\right) \\ \sigma_1^2 = 1 \Big/ \left(\dfrac{1}{\sigma_0^2} + \dfrac{1}{\sigma^2}\right) \end{cases}$

2 Calcul symbolique

On considère maintenant le type de distribution suivant.

```
type dist =
  | Beta of node * node
  | Bernoulli of node
  | Gaussian of node * node
  | V of float

and node = dist ref
```

Les paramètres de chaque distribution sont des références qui peuvent être mises à jour pendant l'inférence. Pour simplifier, on suppose que les valeurs concrètes sont toutes de type `float`.

Question 2. Implémenter les opérateurs probabilistes qui effectuent le calcul symbolique. `sample` associe simplement une variable à une distribution (sans tirer de valeur aléatoire), et `observe d v` met à jour les paramètres de la distribution `d` étant donné l'observation `v`. On lève une exception `Not_tractable` si les calculs symboliques échouent.

```
val sample : dist → node
(* return a new node with distribution [dist] *)

val observe : dist → float → unit
(* update the parameters of the distribution [dist] given an observation
   raise [Not_tractable] if this is not possible. *)

val infer : (α → node) → α → dist
(* return the posterior distribution of a model
   we assume that a model always returns a value of type [node]. *)
```

Nous avons maintenant un moteur d'inférence symbolique qui permet déjà de traiter les exemples de la section 1. Malheureusement, cette implémentation ne fonctionne que pour très peu de modèles.

Question 3. Tester votre code sur les exemples de la section 1.

Question 4. Donner un exemple (simple) de modèle pour lequel l'inférence échoue.

3 Inférence semi-symbolique

Pour traiter plus de cas, l'idée de l'inférence semi-symbolique est de combiner le calcul symbolique avec une méthode particulaire approchée. Chaque particule effectue les calculs symboliques de la Section 2. Si les calculs symboliques échouent on tire aléatoirement des valeurs concrètes pour certaines variables aléatoires avant de continuer l'exécution. La distribution a posteriori est approximée par une *mixture* de distributions $\Sigma_{i=1}^{N} w_i \times d_i$ où N est le nombre de particules, w_i est le poids de la particule i, et d_i est la distribution calculée par la particule i.

Question 5. Implémenter l'opérateur `infer` avec l'algorithme *importance sampling* en utilisant un argument `prob` explicite (vu en cours).

```
type prob : {id: int; scores: float array}
val infer : (prob → α → β) → α → β Distribution.t
```

Notes :

- L'opérateur `infer` est polymorphe et peut être utilisé avec des modèles de type `prob` $\rightarrow \alpha \rightarrow$ `node` qui renvoient une distribution symbolique.
- L'implémentation de `infer` suppose que les opérateurs probabilistes `sample` et `observe` prennent un argument supplémentaire `prob` pour enregistrer le score des particules.

Question 6. Définir un opérateur `value` qui permet d'échantillonner un `node`, i.e., de fixer sa valeur à `V v` où `v` est une valeur tirée aléatoirement dans la distribution du `node` avant de renvoyer la valeur `v`.

```
val value : node → float
```

Question 7. Implémenter les opérateurs `sample` et `observe` pour finir l'implémentation de l'algorithme *importance sampling*. Pour cette question, chaque particule doit renvoyer une distribution `V v`.

```
val sample : prob → dist → node
val observe : prob → dist → float → unit
```

On veut maintenant ajouter le calcul symbolique de la section 2 à notre opérateur `observe`. Pour simplifier, on ne traite que les paires de variable aléatoires conjuguées X, Y où Y est la distribution observée dont l'un des paramètre est X (voir Section 1).

Si les calculs symboliques ne sont pas possibles (X et Y ne sont pas conjuguées) on échantillonne X pour mettre à jour le score de la particule. Sinon (X et Y sont conjuguées) l'évaluation de `observe` Y `v` se décompose en 3 étapes :

- Si X est elle-même une distribution paramétrée, on commence par échantillonner les paramètres de X.
- On calcule la distribution *marginale* de Y qui permet de mettre à jour le score de la particule.
- On met à jour les paramètres de X en fonction de l'observation `v`.

Question 8. Implémenter l'opérateur `observe` avec ce nouveau comportement.

4 Évaluation

Question 9. Proposez quelques modèles simples pour illustrer les forces et les faiblesses de cet algorithme d'inférence.

Question 10. Comparer les performances des 3 algorithmes d'inférence sur ces exemples : symbolique, *importance sampling*, et semi-symbolique.

- Quelle est le surcoût du calcul symbolique pour un nombre de particules fixé ?
- Combien faut-il de particules pour atteindre une précision souhaitée ?

5 Extensions

Question 11. Quel est le problème si on remplace *importance sampling* par un *particle filter* pour l'implémentation de `infer` ? Proposer une solution.

Question 12. Quel est le problème si la valeur de retour d'un modèle n'est pas une variable aléatoire mais une expression, e.g., `let x = sample d in x + 1` ? Proposer une solution.

Question 13. Peut-on éviter d'échantillonner certaines variables pour améliorer les performances de l'algorithme ?

Plusieurs langages de programmation probabilistes exploitent les relations de conjugaisons pour améliorer la précision de l'inférence (Pyro, Augur, Birch, ProbZelus). Le moteur de calcul symbolique présenté ici est très simplifié (on ne traite que des paires de variables aléatoires conjuguées). Il est possible d'utiliser des moteurs symboliques plus puissants (e.g., *Delayed Sampling* ou *Belief Propagation*) au prix de structures de données plus élaborées.